Let's Test with KernelCI

Khouloud Touil, BayLibre

Introductions

BayLibre

- embedded Linux consultancy
- based in Nice, France
- ~40 engineers
- open-source focus
  - top 20 Linux kernel contributor
  - top 5 AGL contributor
  - u-boot, Zephyr, ATF, OP-TEE, Yocto
  - automated testing, SOTA

Khouloud

- SW engineer
- based in Nice, France
- KernelCI contributor

KernelCI

Um, so what is… KernelCI

A free service for **testing** Linux on a huge variety of hardware platforms
https://kernelci.org

An open source **project**
the software behind the service

A **collaboration hub** for hardware-focused, open source testing and automation

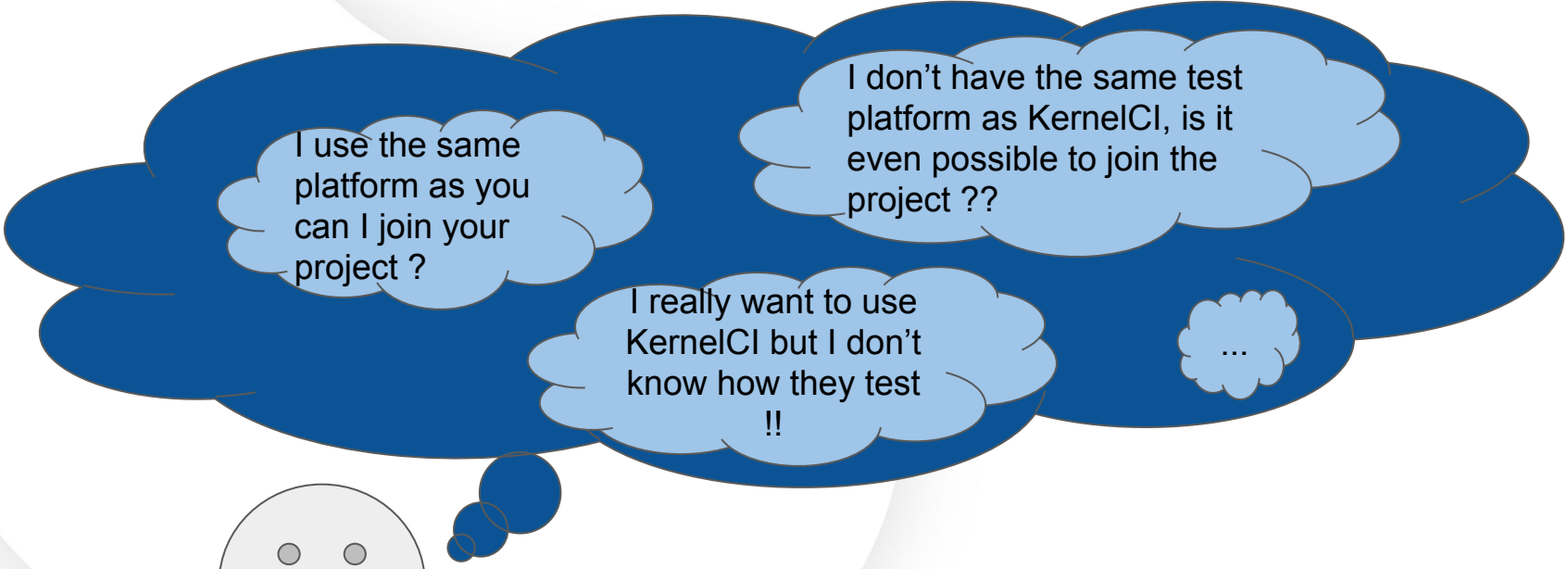
THE LINUX FOUNDATION

Mission Statement

To ensure the quality, stability and long-term maintenance of the Linux kernel by maintaining an open ecosystem around test automation practices and principles.

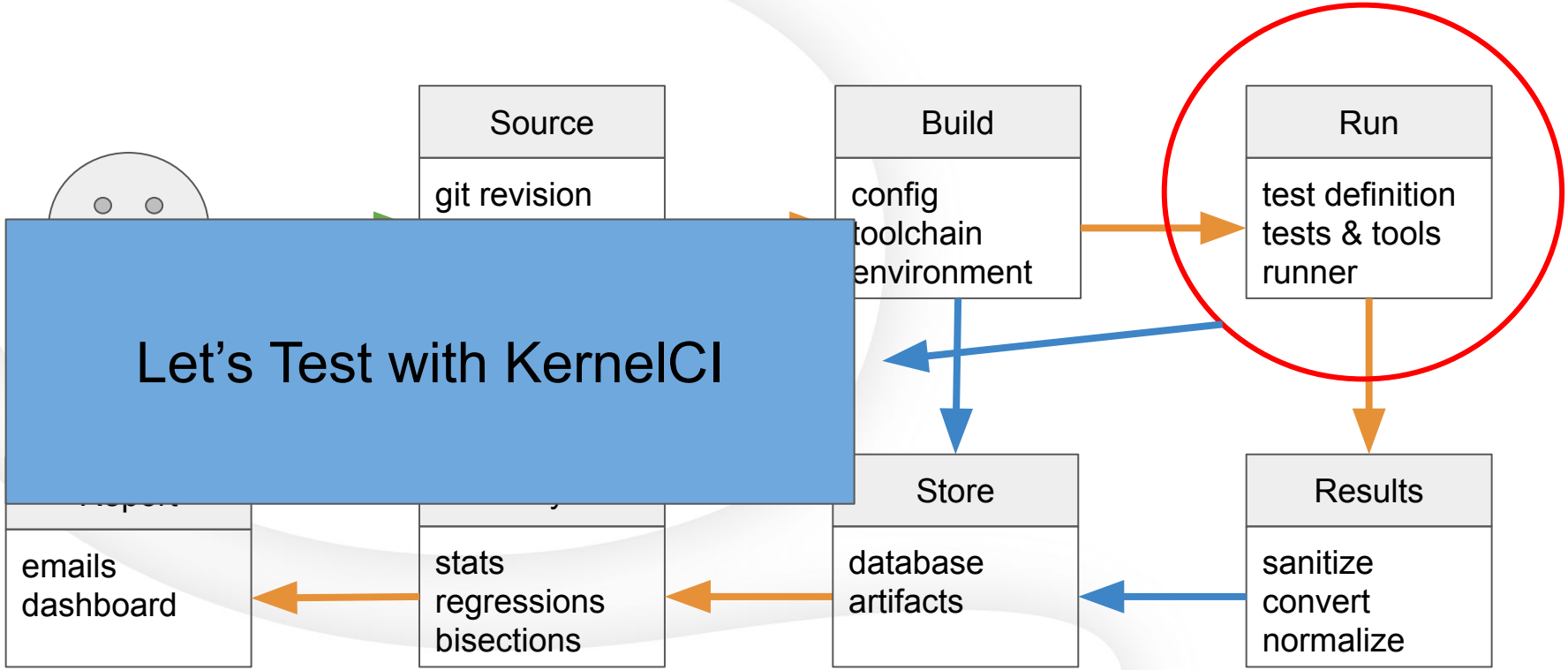
https://foundation.kernelci.org/mission-objectives/

Founding Members



- Kernel Developer
- Kernel Maintainer
- User

Quick Recap: KernelCI Modular Pipeline



https://elinux.org/ELC_2020_Presentations

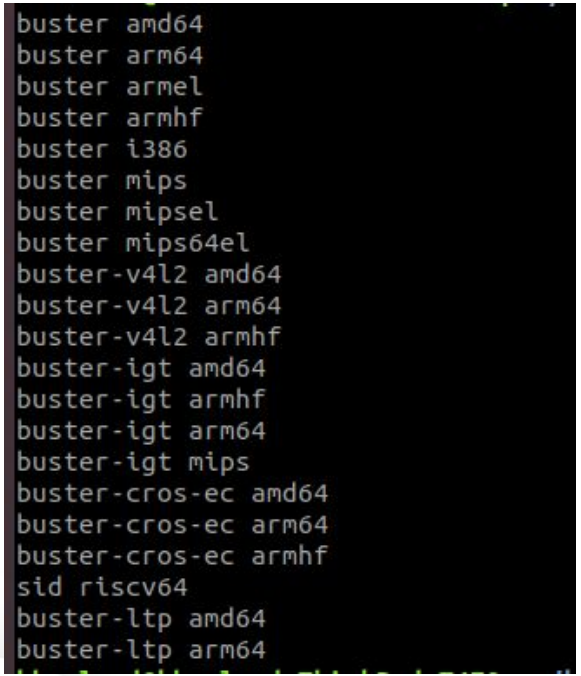
Let's Test with KernelCI

- Prepare the Test suite/case
  - Create the Test plan
  - Customize the rootfs image in the run
  - Add a new rootfs image
- Run the Test
- Submit Test results
  - LAVA users
  - Other users : Never too late to join

Prepare your Test suite/case

- KernelCI provides prebuilt rootfs image that you can use:
  - List the existing rootfs images`**kci_rootfs list_variants**`

- Default rootfs image based on **Debian** : **buster**
  - Arch supported: x86_64, armhf, armel, arm64, mips,riscv,...
  - Using **Debos** to create these images
  - debos project: https://github.com/go-debos/debos



Let's clone kernelci-core

- Link for kernelci-core repo: https://github.com/kernelci/kernelci-core

```
build-configs.yaml  kci_build   kernelci.conf.sample      push-source.py      src
COPYING             kci_data    lab-configs.yaml          README.md           templates
db-configs.yaml     kci_rootfs  lava-v2-callback.py       requirements.txt    test-configs.yaml
doc                 kci_test    Makefile                  rootfs-configs.yaml tests
jenkins             kernelci    push-bisection-results.py setup.py            test-suites
```

- Files/directories:
  - test-configs.yaml
  - rootfs-configs.yaml
  - lab-configs.yaml
  - templates/
  - jenkins/
- Tools
  - kci_rootfs
  - kci_test

Steps to create The "Test suite/case"

- Simple Test case: **echo "Hello world !"**

  - Step one: create relevant directory under `templates` directory
    - mkdir templates/hello-world/

  - Step Two: create you test plan definition
    - Two jinja2 files

```
- test:
    timeout:
      minutes: 5
    definitions:
    - repository:
        metadata:
          format: Lava-Test Test Definition 1.0
          name: hello-world
          description: "hello-world test plan"
          os:
          - oe
```

```
{% extends 'boot/generic-uboot-tftp-ramdisk-boot-template.jinja2' %}
{% block actions %}
{{ super () }}

{% include 'hello-world/hello-world.jinja2' %}

{% endblock %}
```

- Step Three: update the **test_plan** section
  - Modify the **test-configs.yaml** with the new test plan

```
test_plans:

  hello_world:
    rootfs: debian_buster_ramdisk
```

- Step Four: update the **test_configs** section with the test plan for the device type you want to test
  - Modify the **test-configs.yaml** with the new test plan

```
test_configs:

  - device_type: bcm2711-rpi-4-b
    test_plans:
      - hello-world
```

KernelCI

Customize the rootfs image in the run

- More complicated Test than “Hello world !”
  - Keep using a prebuilt rootfs but not enough !!!
  - A need for a customized rootfs ??!!

  - Create a test script (sh,py,..)
    - Customize your rootfs in the test script
    - Execute the test
    - Create yaml file
    - Create the test plan definition: jinja2

```
- test:
    timeout:
      minutes: 5
    definitions:
    - repository: https://github.com/touilkhouloud/kernelci-core
      branch: let-s-test-with-KernelCI
      from: git
      history: False
      path: templates/hello-world-audio/hello-world-audio.yaml
      name: hello-world
```

```
meta
        - functional
params:
  test_params: ""
run:
    steps:
        - ./templates/hello-world-audio/hello-world-audio.sh
```

```
#!/bin/sh

apt-get install w
apt-get install s
apt-get install l
mkdir ~/test
cd ~/test/
wget https://githu
play -q hello-word-audio.mp3
if [ $? -eq 0 ];then
    lava-test-case hello-word-audio --result pass
fi
```

- update the **test_plan** section
  - Modify the **test-configs.yaml** with the new test plan

```
test_plans:

  hello_world-audio:
    rootfs: debian_buster_ramdisk
```

- Update the **test_configs** section with the test plan for the device type you want to test
  - Modify the **test-configs.yaml** with the new test plan

```
test_configs:

  - device_type: bcm2711-rpi-4-b
    test_plans:
      - hello-world-audio
```

Add a new rootfs image

- More sophisticated Test
- Need for a new rootfs image

  - Step One: modify `**rootfs-configs.yaml**` to add your new rootfs definition in the **rootfs_configs** section
    - Arch list
    - extra_packages

  - Step Two: build your rootfs image
    → Using KernelCI tool: **kci_rootfs build** using **debos**

```
rootfs_configs:
  debian-buster-hello-world-audio:
    rootfs_type: debos
    debian_release: buster
    arch_list:
      - arm64
    extra_packages:
      - wget
      - sox
      - libsox-fmt-mp3
```

Add a new rootfs image

- Step Three: update the **file_systems** section in **test-configs.yaml**
  With the new rootfs image

```
file_systems:

  debian_buster_hello_world_audio:
    type: debian
    ramdisk: '{arch}/hello-world-audio/rootfs.cpio.gz'
```

- Step Four: update the **test_plans** section in **test-configs.yaml**
  To use the new rootfs image

```
test_plans:

  hello_world-audio:
    rootfs: debian_buster_hello_world_audio
```

Add a new Test suite rootfs image

- Things getting complicated !!
- Tests are more complicated than `**hello world !**` !!
  - Let's build our rootfs with our Test

    - Add the **config script**: "script/buster-igt.sh"
      → Add the build Test part in the script
      → Runs at rootfs creation time
      → Add non debian package tools

    - Add a **test_overlays**: **overlays/igt**
      →"jenkins/debian/debos/overlays/igt/usr/bin/igt-parser.sh"
      → **igt-parser.sh**: script to run & parse results in LAVA style

```
buster-igt:
  rootfs_type: debos
  debian_release: buster
  arch_list:
      - amd64
      - armhf
      - arm64
      - mips
  extra_packages:
      - libcairo2
      - libdw1
      - libglib2.0-0
      - libpciaccess0
      - libprocps7
      - libudev1
      - libunwind8
  extra_packages_remove:
      - bash
      - e2fslibs
      - e2fsprogs
      - fonts-dejavu-core
      - klibc-utils
      - libext2fs2
```

```
- test:
    timeout:
      minutes: 10
    definitions:
    - repository:
        metadata:
          format: Lava-Test Test Definition 1.0
          name: {{ plan }}
          description: "IGT test plan"
          os:
          - oe
          scope:
          - functional
        run:
          steps:
          - >
              IGT_FORCE_DRIVER={{ drm_driver }} /usr/bin/igt-parser.sh
              {{ tests|wordwrap(1, False, "\n                ") }}
      from: inline
      name: {{ plan }}
      path: inline/{{ plan }}.yaml
      lava-signal: kmsg
```

Let's run the Test

- Not yet ….! Let's generate the Test first !!
  - LAVA requirements & setup
    - Setup the lab physically or QEMU
    - Create a user and a token
    - Add your lab to the list of labs in **lab-configs.yaml** along with the list of test plans to be run
    - KernelCI team will generate for you a lab **token** that permit you to submit your results after the test is done ✅

```
labs:

  lab-baylibre:
    lab_type: lava
    url: 'https://lava.baylibre.com/RPC2/'
    filters:
      - blocklist: {tree: [drm-tip]}
      - passlist:
          plan:
            - hello-world-audio
```

- Use **kci_test generate** tool to generate the Test job definition: job.yaml

```
./kci_test generate --storage $STORAGE --bmeta-json bmeta.json\
                    --dtbs-json dtbs.json --lab-config $LAB --plan $PLAN\
                    --output $LAB  --user $USER  --lab-token $TOKEN
```

- Finally running the Test using **kci_test submit**

```
./kci_test submit --lab-config $LAB --user $USER\
                  --lab-token $TOKEN --jobs job.yaml
```

```
usage: kci_test generate [-h] [--bmeta-json BMETA_JSON] [--storage STORAGE]
                         [--lab-config LAB_CONFIG] [--plan PLAN]
                         [--target TARGET] [--output OUTPUT]
                         [--dtbs-json DTBS_JSON] [--lab-json LAB_JSON]
                         [--user USER] [--lab-token LAB_TOKEN]
                         [--db-config DB_CONFIG] [--callback-id CALLBACK_ID]
                         [--callback-dataset CALLBACK_DATASET]
                         [--callback-type CALLBACK_TYPE]
                         [--callback-url CALLBACK_URL] [--mach MACH]

optional arguments:
  -h, --help            show this help message and exit
  --bmeta-json BMETA_JSON
                        Path to the build.json file
  --storage STORAGE     Storage URL
  --lab-config LAB_CONFIG
                        Test lab config name
  --plan PLAN           Test plan name
  --target TARGET       Name of a target platform
  --output OUTPUT       Path the output directory
  --dtbs-json DTBS_JSON
```

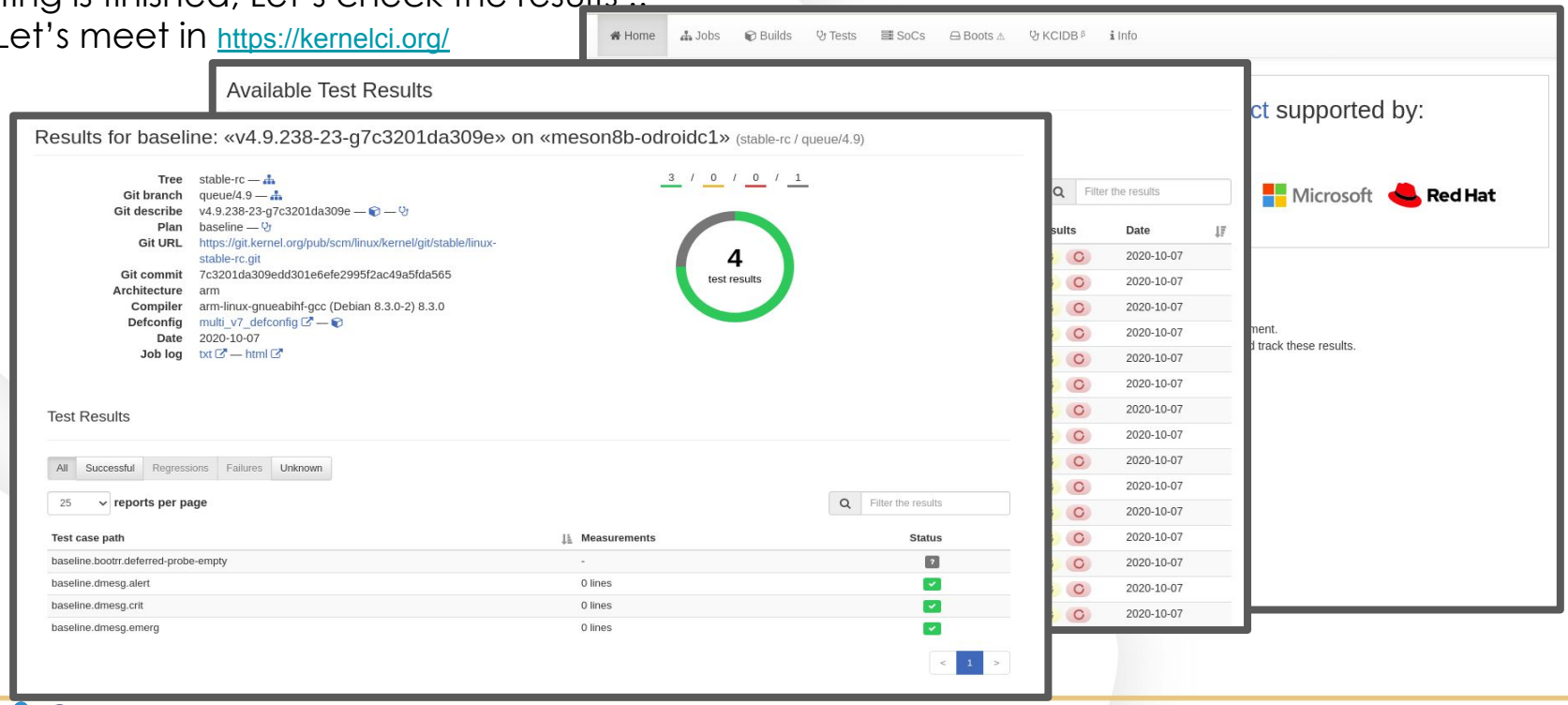
```
usage: kci_test submit [-h] [--lab-config LAB_CONFIG] [--user USER]
                       [--lab-token LAB_TOKEN] [--jobs JOBS]

optional arguments:
  -h, --help            show this help message and exit
  --lab-config LAB_CONFIG
                        Test lab config name
  --user USER           Test lab user name
  --lab-token LAB_TOKEN
                        Test lab token
  --jobs JOBS           File pattern with jobs to submit
```

KernelCI

Submit Test results- LAVA users

→ Testing is finished, Let's check the results !!
Let's meet in https://kernelci.org/



KernelCI

→ Too lazy to come to us, we will meet then in your email box !!

stable-rc/queue/4.9 baseline: 96 runs, 1 regressions (v4.9.238-26-g1959353b3c5c)

KernelCI bot

stable-rc/queue/4.9 baseline: 96 runs, 1 regressions (v4.9.238-26-g1959353b3c5c)

Regressions Summary
---

platform	arch	lab	compiler	defconfig	results
qemu_i386-uefi	i386	lab-collabora	gcc-8	i386_defconfig	0/1

Details: https://kernelci.org/test/job/stable-rc/branch/queue%2F4.9/kernel/v4.9.238-26-g1959353b3c5c/plan/baseline/

Test: baseline
Tree: stable-rc
Branch: queue/4.9
Describe: v4.9.238-26-g1959353b3c5c
URL: https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable-rc.git
SHA: 1959353b3c5c434342156f836e0a889c5e22fd39

Test Regressions
---

platform	arch	lab	compiler	defconfig	results
qemu_i386-uefi	i386	lab-collabora	gcc-8	i386_defconfig	0/1

Details: https://kernelci.org/test/plan/id/5f7dd56b8b761ca0474ff3e0

Results: 0 PASS, 1 FAIL, 0 SKIP
Full config: i386_defconfig
Compiler: gcc-8 (gcc (Debian 8.3.0-6) 8.3.0)
Plain log: https://storage.kernelci.org//stable-rc/queue-4.9/v4.9.238-26-g1959353b3c5c/i386/i386_defconfig/gcc-8/lab-collabora/baseline-qemu_i386-uefi.txt
HTML log: https://storage.kernelci.org//stable-rc/queue-4.9/v4.9.238-26-g1959353b3c5c/i386/i386_defconfig/gcc-8/lab-collabora/baseline-qemu_i386-uefi.html
Rootfs: http://storage.kernelci.org/images/rootfs/buildroot/kci-2020.05-3-g27eeeac7da2d/x86/baseline/rootfs.cpio.gz

* baseline.login: https://kernelci.org/test/case/id/5f7dd56b8b761ca0474ff3e1
new failure (last pass: v4.9.238-23-g7c3201da309e)

KernelCI

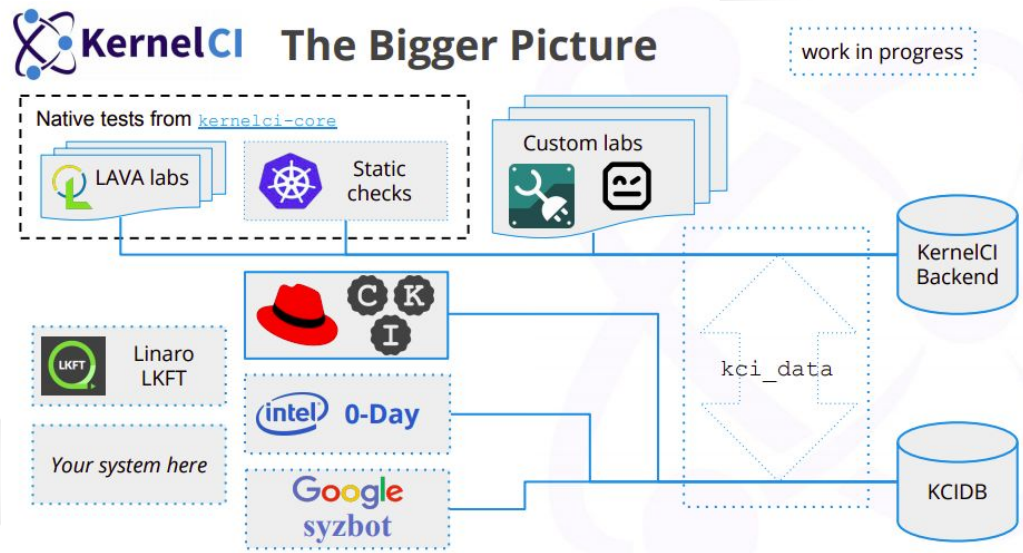
Still want to join KernelCI ...

KernelCI

KCIDB

KCIDB tool

- It's a tool for submitting and querying Linux Kernel CI reports



→ kcidb Grafana

https://linuxplumbersconf.org/event/7/contributions/730/attachments/505/1165/Unifying_Test_Reporting_with_KernelCI.pdf

Origin	Description	Output files	Log	Status	Tests
kernelci	v5.9-rc8-157-ge0fd7bf13452	Name: modules, System.map	File	✔	FAIL

Start time	Duration	Architecture	Configuration
2020-10-08 00:21:18.348+00	00:04:43.780	arm64	defconfig+kselftest

Command

Compiler

aarch64-linux-gnu-gcc (Debian 8.3.0-2) 8.3.0

Tests

Origin	Path	Description	Start time	Duration	Status
kernelci	baseline.dmesg.crit	baseline on qemu_arm64-virt-gicv3-uefi in lab-baylibre	2020-10-08 00:21:19.762+00		PASS
kernelci	baseline.dmesg.alert	baseline on qemu_arm64-virt-gicv3-uefi in lab-baylibre	2020-10-08 00:21:19.766+00		PASS
kernelci	baseline.dmesg.emerg	baseline on qemu_arm64-virt-gicv3-uefi in lab-baylibre	2020-10-08 00:21:19.768+00		PASS
kernelci	baseline.bootrr.deferred-probe-empty	baseline on qemu_arm64-virt-gicv3-uefi in lab-baylibre	2020-10-08 00:21:19.774+00		PASS
kernelci	baseline.login	baseline on qemu_arm64-virt-gicv3 in lab-cip	2020-10-08 00:21:27.214+00		PASS
kernelci	baseline.dmesg.crit	baseline on qemu_arm64-virt-gicv3 in lab-cip	2020-10-08 00:21:27.220+00		PASS
kernelci	baseline.dmesg.alert	baseline on qemu_arm64-virt-gicv3 in lab-cip	2020-10-08 00:21:27.223+00		PASS
kernelci	baseline.dmesg.emerg	baseline on qemu_arm64-virt-gicv3 in lab-cip	2020-10-08 00:21:27.226+00		PASS
kernelci	baseline.bootrr.deferred-probe-empty	baseline on qemu_arm64-virt-gicv3 in lab-cip	2020-10-08 00:21:27.232+00		PASS
kernelci	baseline.bootrr.deferred-probe-empty	baseline on meson-gxl-s905x-libretech-cc in lab-clabbe	2020-10-08 00:21:29.818+00		PASS
kernelci	baseline.dmesg.crit	baseline on meson-g12a-u200 in lab-baylibre	2020-10-08 00:21:36.916+00		PASS
kernelci	baseline.dmesg.alert	baseline on meson-g12a-u200 in lab-baylibre	2020-10-08 00:21:36.918+00		PASS
kernelci	baseline.dmesg.emerg	baseline on meson-g12a-u200 in lab-baylibre	2020-10-08 00:21:36.920+00		PASS
kernelci	baseline.bootrr.deferred-probe-empty	baseline on meson-g12a-u200 in lab-baylibre	2020-10-08 00:21:36.928+00		PASS

KernelCI

Origin	Description	Output files	Log	Status	Tests
redhat		Name: kernel.tar.gz	File	✔	PASS

Start time	Duration	Architecture	Configuration
2020-03-27 22:33:00.094+00	00:03:18.000	x86_64	fedora

Command

make -j30 INSTALL_MOD_STRIP=1 targz-pkg

Compiler

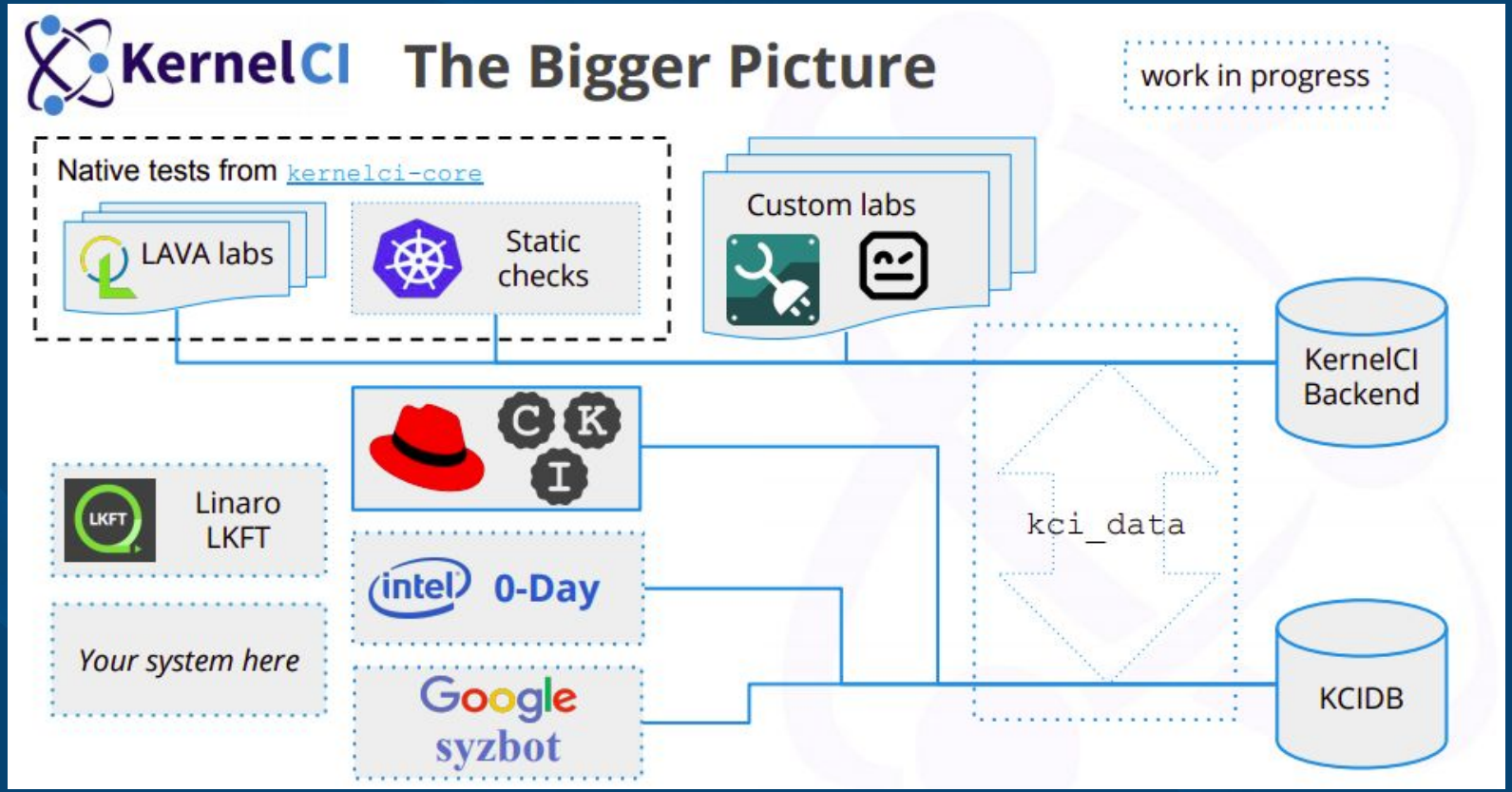
gcc (GCC) 9.2.1 20190827 (Red Hat 9.2.1-1)

Tests

Origin	Path	Description	Start time	Duration	Status
redhat	boot	Boot test	2020-03-27 23:05:51+00	00:05:01.000	PASS
redhat	redhat_bridge	Networking bridge: sanity	2020-03-27 23:10:52+00	00:05:56.000	PASS
redhat	redhat_ethernet	Ethernet drivers sanity	2020-03-27 23:16:48+00	00:00:14.000	PASS
redhat	redhat_macsec	Networking MACsec: sanity	2020-03-27 23:17:02+00	00:01:30.000	PASS
redhat	redhat_socket_fuzz	Networking socket: fuzz	2020-03-27 23:18:32+00	00:17:00.000	PASS
redhat	redhat_sctp_auth_sockopts	Networking sctp-auth: sockopts test	2020-03-27 23:35:32+00	00:00:37.000	PASS
redhat	redhat_igmp	Networking: igmp conformance test	2020-03-27 23:36:09+00	00:09:11.000	PASS
redhat	redhat_pmtu	Networking route: pmtu	2020-03-27 23:45:21+00	00:04:36.000	PASS
redhat	redhat_route	Networking route_func - local	2020-03-27 23:49:57+00	00:05:06.000	PASS
redhat	redhat_route	Networking route_func - forward	2020-03-27 23:55:03+00	00:05:02.000	PASS
redhat	redhat_tcp_keepalive	Networking TCP: keepalive test	2020-03-28 00:00:05+00	00:02:04.000	PASS
redhat	redhat_udp_socket	Networking UDP: socket	2020-03-28 00:02:09+00	00:01:02.000	PASS
redhat	redhat_geneve	Networking tunnel: geneve basic test	2020-03-28 00:03:11+00	00:04:24.000	PASS
redhat	redhat_gre	Networking tunnel: gre basic	2020-03-28 00:07:35+00	00:06:02.000	PASS
redhat	redhat_vxlan	Networking tunnel: vxlan basic	2020-03-28 00:13:37+00	00:05:56.000	PASS

KernelCI

Recap



https://linuxplumbersconf.org/event/7/contributions/730/attachments/505/1165/Unifying_Test_Reporting_with_KernelCI.pdf

Getting Involved

Blog: https://foundation.kernelci.org/blog/

Twitter: https://twitter.com/kernelci (#kernelci, @KernelCI)

Mailing list: https://groups.io/g/kernelci/topics

IRC: #kernelci on Freenode

Weekly technical call: Tuesdays: 16h-17h UTC

LinkedIn: https://www.linkedin.com/company/kernelci-org/

KernelCI

Photo Credits

Mixing ingredients: https://flic.kr/p/2jBQqYv

House image: https://flic.kr/p/8o21aj

Lazy cat: https://flic.kr/p/HU2VzC

KernelCI